Why ?

Reason for and technical solutions in the project

by John-Eric Söderman

Table of Contents

1 In the beginning

In this document we try to explain the question “Why ?”. To use Y2018-Text project you have to read the HTML of the project Regular Expression in ADA, Y2018-Text-project (https://hellvik.github.io/)

We try to build a base for Regular Expression handling independent of any natural language as presented by the work done by Unicode.org but there are limits.
The list of codepoints extends for every version and in the project are tools for extending list of defined codepoints. It seems that Unicode.org still has much work to do to reach the target of including all form of scripts into the standard.

In this document we try to reserve the words ‘character’ and ‘string’ for ADA Character (a byte which is a part of UTF-8 form byte group) and ADA String, a byte array containing ADA Characters. The result is that we use the terms *codepoint* and *codepoint array* (or the type name CFix) as a replacement for character and string when we refer to these 21-bit entities or arrays.

The use of Codepoint, CFix and CVar (a variable length CFix type) is presented at Regular Expression in ADA, Y2018-Text-project project pages. This project has no support for UTF-16 or UTF-32 and suggest that these ‘standard types’ (Wide_String, Bounded_Wide_String, Unbounded_Wide_String, Wide_Wide_String, Bounded_Wide_Wide_String, Unbounded_Wide_Wide_String) is not used in ADA code using this project if possible. About UTF-8 ‘standard types’ (Bounded_String, Unbounded_String) has the same problems and the same suggestion applies don’t use if possible[1]. It seems that the support in ADA has a huge amount of errors and inconsistency in the specification of these types.

The target for Regular Expression in ADA, Y2018-Text-project is an implementation of Regular Expression and the most interesting part can be found in the subchapter The Daisy chain (page Error: Reference source not found) where we use multi threading feature of ADA as a solution. On a multi-threading platform this maybe a faster solution than an old single thread solution, but is has it’s drawbacks.

A change to the original implementation of Regular Expression in Perl language is that we don’t return the result as codepoint-arrays (or strings in Perl) but we return index pairs (Q_A type) of the codepoint-array which was the target for the search (page 17).

All packages are sub-packages to Y2018

```
package Y2018 is
```

1 The problematic definition is *function "&" (Left : in String; Right : in Unbounded_String) return Unbounded_String* and *function "&" (Left : in String; Right : in Bounded_String) return Bounded_String* which makes it impossible to join a string to a string from left to right without parentheses and casting. To make it worse, it is in the standard and cannot be changed without loosing upward compatibility

```
      Version:constant String:="2";
end Y2018;
```

---

and to Y2018.Text packages

```
package Y2018.Text is
      Version:constant String:=Y2018.Version & ".5.0";
end Y2018.Text;
```

The target is a natural language independent solution for text handling based on the work of Unicode.org. There exists problem areas

- ✔ Title case is not language independent. A change from lower case or upper case cannot be done to title case if you don't know in which language the text is written[2]
- ✔ Turkish and Azerbaijan uses a casing of i-letter in a way that is not supported here[3]
- ✔ Case conversion from one codepoint to many is not supported[4]
- ✔ Definition of a word in Western languages varies and word-support is only partly supported through the [\w] pattern marker[5]
- ✔ Possibility of construction of a codepoint from more than one is not supported. The rule is that one codepoint (21-bits) is one Unicode.org codepoint entity[6]

Above some of the limits in Latin based scripts, as an example, in our efforts to handle text not knowing which language the text is written in.

We are not repeating information from Regular Expression in ADA, Y2018-Text-project pages but to enlarge and explain why things works as described on the project pages.

2 In german all nouns starts with titlecase/ uppercase codepoint but how do we know if the text is german and the word is a noun ?

3 Uppercase I converts to lowercase ı-letter (lowercase dotless i) and uppercase İ-letter (uppercase I with dot above) to i-letter (i-letter, with dot above)

4 In Unicode.org standard it is possible to construct a codepoint from more than one codepoint

5 This has been a problem solved in telephone books in names with prefixes (von, af)

6 In swedish writing some sounds of the spoken language is written with a combination of two or three letters and varying depending on if it is present, past or future tense. We are not talking about 'endings'.

2 Text in ADA

ADA (2012) don’t support Unicode characters although UTF-8 is supported but …
In this project we enlarge support for Unicode characters as Unicode charters or as Unicode.org calls them CODEPOINT’s. This ‘enlargement’ is done only for literals by introducing two entities in the language

(1) “XYZ”c – corresponding to a constant Codepoint array
(2) ‘Y’c – corresponding to a constant Codepoint value

This solution is not supported even in ADA 2022 and maybe in the future something like it will be in ADA, but before this happens code with these entities must be preprocessed (page 26) so that these entities can be replaced with valid ADA literals.

3 Core

To make this type of Regular Expression possible a base package is needed, containing

- support for Codepoint (21-bit entity)
- fixed and variable array like support for Codepoint (CFix & CVar)
- conversion between Character (UTF-8) and Codepoint
- and some added features
  - hexadecimal support for Codepoint arrays
  - correct overloading of &-operator between Character, CFix and CVar
  - overloading of relational operators i.e. =, <, >, <= and >=

```
package Y2018.Text.Core is
  type TRINITY is (TRI_TRUE,TRI_FALSE,TRI_UNDEF);
  type CodePoint is range 0 .. 2097151;
    for CodePoint'Size use 21;
  type CFix is Array(Positive range <>) of CodePoint;
  CodePoint_Space: constant CodePoint :=CodePoint'Val(16#20#);
  CodePoint_Replacement_Character: constant CodePoint:=CodePoint'Val(16#FFFD#);
  Null_CFixX : CFix(1 .. 0); -- used only for defining Null_CFix as a constant
  Null_CFix: constant CFix := Null_CFixX;
  CFix_Space:constant CFix:=(1=>CodePoint'Val(16#20#));
end Y2018.Text.Core;
```

Mostly the support is done through following sub packages:

- definition y2018-text-core-cvarpack.ads and the body y2018-text-core-cvarpack.adb
- definition y2018-text-core-str.ads and the body y2018-text-core-str.adb
- definition y2018-text-core-utf.ads and the body y2018-text-core-utf.adb

As a base we need static tables Y2018.Text.Core.C32 (table for converting 8 to 21 bit), Y2018.Text.Core.C8 (table for converting 21 to 8 bit), Y2018.Text.Core.GC (tables for connecting General Code value to codepoint) and Y2018.Text.Core.Ultcase (table for converting between upper, lower and title case). Y2018.Text.Core.GC and Y2018.Text.Core.Ultcase changes from Unicode version to newer version.

4 Jets

Package Y2018.Text.Jets is the base for Regular Expression handling in the project.

```
package Y2018.Text.Jets is
  type I_A is array (1..2) of Integer;
  type I_A_ARRAY is Array(Integer range <>) of I_A;
      -- < more Constants > --
  type MCHAR_AR is Array(Integer range <>) of CodePoint;
  type REQ_TY is (REQ_BEGIN, REQ_CHAR, REQ_PERIOD, REQ_CLASS_ELEM, REQ_CLASS,
    REQ_PALCAP, REQ_PAL, REQ_UPDOWN, REQ_CLASS_CHAR, REQ_CLASS_RANGE,
    REQ_CLASS_GC, REQ_CLASS_GCU, REQ_CLASS_SET, REQ_CLASS_OTHER,
    REQ_POS, REQ_NEG, REQ_NULL);
  function subIA(value:CFix;a:I_A) return CFix;
  function subIA(value:String;a:I_A) return String;
  function subIA(value:CVar;a:I_A) return CVar;
  function To_String(ia:I_A) return String;
end Y2018.Text.Jets;
```

Use of defined types

- I_A defines a range usually in a CFix or CVar array but is also used for Ada Strings.
- I_A_Array used in returning the result of a successful Regular expression search
- REQ_TY contains the type values of a pattern compilation

Two basic function is defined here

- subIA – a sub-range of a CFix, CVar or String. Note the range is always in 21-bit codepoint entities.
- To_String – returns the I_A value nicely for test purposes

Sub-packages are

- RangeVectorPack (page 8) – uses only the Core package of Y2018-Text
- JetsTool (page 9) – uses RangeVectorPack
- MatchPack (page 14) – uses RangeVectorPack and JetsTool
- Compiler (page 10) – uses RangeVectorPack, JetsTool and MatchPack
- PatternPack (page 14) – uses all above

This is the hierarchy of the packages in Jets and a base for the compilation of Jets and it's sub-packages.

4.1 RangeVectorPack

RangeVectorPack main use is to keep track of groups of CodePoints. A group is defined with Velem-type of codepoint-values from first (f) to last (l). Group cannot contain values from another group and if a group with last value plus one is the first value of next group they are joined to one group or to one Velem entity.

```
type Velem is
record
  f: Integer;
  l: Integer;
end record;

package Set_TY is new Ada.Containers.Ordered_Sets(
  Element_type => Velem,
  "<" => less,
  "=" => equal);
```

```
private type RangeVector is tagged
record
  v: Set_TY.Set;
end record;
```

To handle values in a RangeVectorPack variable (not all):

- putC – insert a range, from codepoint value first (f) to last (l), or insert a specific codepoint value
- putGC – insert all values from an Unicode.org general class
- putSET – update values from a Map_TY, putSET is a way to include contents in a map of Set's (Map_TY variable)
- exists – check if this codepoint value is in the RangeVectorPack (or Set)

RangeVectorPack is the implementation of codepoint groups in a pattern. RangeVectorPack has it's own private type RangeVector which stores the ordered set.

Defined in RangeVectorPack but ..

```
package Map_TY is new Ada.Containers.Ordered_Maps(
  Key_Type=> Y2018.Text.Core.CVarPack.CVar,
  Element_Type=> Set_TY.Set,
  "<"=> lessUR,
  "="=> equalUR);
```

a way to include codepoint groups from outside into RangeVectorPack sets (page 23) using procedure putSet.

4.2 JetsTool

Containing different utilities for the Y2018.Text.Jets package group.

4.2.1 The three of nodes as the result of compiling the Regular expression pattern

A pattern is compiled (see page 10) and as the result we get a three of RE_Elem nodes. Every node is identified through a unique ID of type RE_ID. All nodes are allocated in the heap and must be freed by a call of procedure RE_ELEM_Free.

```
subtype RE_ID is  Integer range 0 .. Integer'Last;
NO_RE_ID: constant RE_ID := 0;
FIRST_RE_ID: constant RE_ID := 1;
type RE_Elem; – Note defined later
type RE_Elem_AC is access RE_Elem;
type RE_ARR is array(Integer range <>) of RE_Elem_AC;
package CHAR_PERIOD_CLASS_Vector_TY is new
  Ada.Containers.Vectors(Natural,RE_Elem_AC);
```

All nodes has also a type, "REQ_TY" and this type is defined in Jest package and contains following values: REQ_BEGIN, REQ_CHAR, REQ_PERIOD, REQ_CLASS_ELEM, REQ_CLASS, REQ_PALCAP, REQ_PAL, REQ_UPDOWN, REQ_CLASS_CHAR, REQ_CLASS_RANGE, REQ_CLASS_GC, REQ_CLASS_GCU, REQ_CLASS_SET,

REQ_CLASS_OTHER, REQ_POS, REQ_NEG, REQ_NULL. Where value REQ_NULL is node type not used.
From every node there exists two links

- left – backward link, or a null
- right – forward link, or a null

for some nodes (REQ_PALCAP, REQ_PAL, REQ_UPDOWN, REQ_POS, REQ_NEG) there is also

- up – up and backward
- down – down and forward

The type of the root node is always REQ_BEGIN, and left-link and up-link is null.
Some node types exists only during the compilation and are removed after compilation (REQ_CLASS_ELEM, REQ_CLASS_CHAR, REQ_CLASS_GC, REQ_CLASS_GCU, REQ_CLASS_SET, REQ_CLASS_OTHER) and a node of type REQ_CLASS replaces these nodes.
To construct a node in the node-three following functions are used

- function CREATE_RE(id:in out RE_ID;reqv: REQ_TY) return RE_ELEM_AC
- function CREATE_RE(id:in out RE_ID;reqv: REQ_TY;vleft:RE_Elem_AC) return RE_ELEM_AC – update node in vleft with a (right-) linkvalue to this node and set (left-) link to point to the node of ‘vleft’.
- function CREATE_RE(id:in out RE_ID;reqv: REQ_TY;vleft:RE_Elem_AC;vup:RE_Elem_AC) return RE_ELEM_AC - update node in vleft with a (right-) linkvalue to this node and set (left-) link to point to the node of ‘vleft’. Update also node in vup with a (down-) linkvalue to this node and set (up-) link to point to the node of ‘vup’.

The CHAR_PERIOD_CLASS_Vector construction is described in chapter Compiler stage 3 the Right Cascade process (page 10) and use in chapter The real matcher, the MatchesI function (page 16).

4.3 Compiler

This package contains the compiler of the pattern in a regular expression (don’t mistake it with the preprosessor CCompile ,page 26).
The call is following procedure:
procedure compileP (s : CFix; urv: RangeVectorPack.Map_TY.Map := RangeVectorPack.Map_TY.Empty_Map ; ptc : out CompileResult)
where

- s: CFix – Pattern codepoint-array which is to be compiled to a node three
- urv: RangeVectorPack.MAP_TY – containing codepoint groups which may or may not be referenced in the pattern
- pct: out CompileResult – the result of the compilation

The ‘pct’, an out structure, has following type:

```
type CompileResult is
record
  state: PST_TY;
  top: RE_Elem_AC; - - link to root node
  current:RE_ID;
  map:MatchPack.MatchID2RE_ID_TY.Map;
    -- mapping of MatchID's to RE id's. Key is MatchID.
  pool:JetsTool.Pool_TY;
end record;
```

The compiler works in stages

1) read through the pattern and create temporary node three
2) replace to nodes with REQ_CLASS construction for those nodes which should not be in a node three
3) read through the node three for updating of CHAR_PERIOD_CLASS vectors. Used for determine which pattern entity is possible first, the Right Cascade process (page 11).

4.3.1 Right Cascade process

The process starts with the call **rightCascade(ptc.top,0)** . The target is to find all possibly starting codepoints in elements of a regular expression where there is possibly more than one alternative. The process is based on

1) rightCascade procedure - rightCascade(currRE:RE_Elem_AC;depth:Integer:=0)
2) downCascade procedure - downCascade(currRE:RE_Elem_AC;depth:Integer:=0)
3) leftUpdate boolean function - leftUpdate(currRE:RE_Elem_AC;valueRE:RE_Elem_AC;depth:Integer:=0)
4) upUpdate procedure - upUpdate(currRE:RE_Elem_AC;valueRE:RE_Elem_AC;depth:Integer:=0)
5) downUpdate boolean function - downUpdate(currRE:RE_Elem_AC;valueRE:RE_Elem_AC;depth:Integer:=0)

The ‘depth’ value is only a support for testing of the process.

4.3.1.1 rightCascade

In rightCascade we go through the note three and search for nodes of type REQ_CLASS, REQ_PERIOD and REQ_CHAR as can be seen from the code:

```
this:=currRE;
while this/=null loop
  case this.v is
  when REQ_PAL => downCascade(this,depth + 1);
  when REQ_PALCAP => downCascade(this,depth + 1);
  when REQ_BEGIN => null;
  when REQ_CLASS => if this.left /= null then result:=leftUpdate(this.left,this,depth + 1);end if;
  when REQ_CHAR => if this.left /= null then result:=leftUpdate(this.left,this,depth + 1);end if;
  when REQ_PERIOD => if this.left /= null then result:=leftUpdate(this.left,this,depth + 1);end if;
  end case;
this:=this.right;
end loop;
```

Although we are not interested in the ‘result’.

4.3.1.2 downCascade

In downCascade we go through the note three but only those parts delegated by the rightCascade and search for nodes of type REQ_PAL or REQ_PALCAP:

```
begin
case currRE.v is
when REQ_PAL =>
  that:=currRE.pal_down;
  while that/=null loop
    if that.right/=null then rightCascade(that.right,depth + 1);end if;
    that:=that.updown_down;
```

```
  end loop;
when REQ_PALCAP =>
  that:=currRE.palcap_down;
  while that/=null loop
    if that.right/=null then rightCascade(that.right,depth + 1);end if;
    that:=that.updown_down;
  end loop;
end case;
```

Although we are not interested in the ‘result’.
Here we have two loops going through the vertical chain and if right link is found calling rightCascade.

4.3.1.3 leftUpdate

This is the ‘kernel’ of the process, here we append a node to CHAR_PERIOD_CLASS_Vector for a alternate start. Only nodes of type REQ_CLASS, REQ_CHAR and REQ_PERIOD can get their vectors appended.

```
 function leftUpdate(currRE:RE_Elem_AC;valueRE:RE_Elem_AC;depth:Integer:=0)
return Boolean is
  this:RE_Elem_AC;
  -- returns TRUE if the search uppwards/left is to continue
  -- returns FALSE if no search upwards/left should be done
 begin
  this:=currRE;
  while this/=null loop
   case this.v is
    when REQ_UPDOWN => upUpdate(this.updown_up,valueRE,depth + 1);
    when REQ_PAL =>
     if this.pal_down = null then null;
     elsif downUpdate(this.pal_down,valueRE,depth + 1) = FALSE then
      return FALSE;
     else null;
     end if;
    when REQ_PALCAP =>
     if this.palcap_down = null then null;
     elsif downUpdate(this.palcap_down,valueRE,depth + 1) = FALSE then
      return FALSE;
     else null;
     end if;
    when REQ_BEGIN => null;
    when REQ_CLASS =>
     if this.class_min = this.class_maxORG then null;
     elsif this.class_greedy then null;
     else
      CHAR_PERIOD_CLASS_Vector_TY.Append(this.char_period_class_vector,valueRE);
     end if;
     if this.class_min > 0 then return FALSE; end if;
    when REQ_CHAR =>
     if this.char_min = this.char_maxORG then null;
     elsif this.char_greedy then null;
     else
      CHAR_PERIOD_CLASS_Vector_TY.Append(this.char_period_class_vector,valueRE);
     end if;
     if this.char_min > 0 then return FALSE; end if;
    when REQ_PERIOD =>
     if this.period_min = this.period_maxORG then null;
     elsif this.period_greedy then null;
```

```
     else
      CHAR_PERIOD_CLASS_Vector_TY.Append(this.char_period_class_vector,valueRE);
     end if;
     if this.period_min > 0 then return FALSE; end if;
   end case;
   this:=this.left;
  end loop;
  return TRUE;
 end leftUpdate;
```

4.3.1.4 upUpdate

Return back to the top of the vertical chain, and call leftUpdate:

```
loop
  case that.v is
  when REQ_UPDOWN => that:=that.updown_up;
  when others =>
   if that.left /= null then result:=leftUpdate(that.left,valueRE,depth + 1);end if;
   exit;
  end case;
end loop;
```

4.3.1.5 downUpdate

Go through the vertical chain from top to bottom and return any findings.

```
 function downUpdate(currRE:RE_Elem_AC;valueRE:RE_Elem_AC;depth:Integer:=0)
return Boolean is
  that:RE_Elem_AC;
  this:RE_Elem_AC;
  cntTRUE:Integer:=0; -- TRUE if the search uppwards/left is to continue
  cntFALSE:Integer:=0;-- FALSE if no search upwards/left should be done
  function getLastOfRight(currRE:RE_Elem_AC) return RE_Elem_AC is
   this:RE_Elem_AC;
  begin
   this:=currRE;
   while this.right /= null loop this := this.right; end loop;
   return this;
  end getLastOfRight;
 begin
  that:=currRE;
  while that/=null loop
   if that.right/=null then
    this:=getLastOfRight(that.right); ← return back to vertical chain
    if leftUpdate(this,valueRE,depth + 1) then cntTRUE:=cntTRUE + 1;
    else cntFALSE:=cntFALSE + 1;
    end if;
   end if;
   that:=that.updown_down;
  end loop;
  if cntTRUE = 0 then return FALSE; -- halt the search
  else return TRUE;-- at least one thread returns TRUE for upwards/left search
  end if;
 end downUpdate;
```

4.4 MatchPack

As for all packages in Y2018-Text project with a name ending in ’Pack’ then private type is key building block, in this case it is Match_TY.

```
private type Match_TY is tagged
record
  map:MatchID2RE_ID_TY.map;
  id:MatchID;
end record;
```

The ‘map’ is an ordered map with link to every node in the search, but also containing the index pair to the codepoint-array where the found result exists. As key is the id of the node.

```
package MatchID2RE_ID_TY is new Ada.Containers.Ordered_Maps(
  Key_Type=> MatchID,
  Element_Type=> MatchID2RE_ID_Elem_TY,
  "<" => MatchID2RE_ID_less,
  "=" => MatchID2RE_ID_equal);
- - - and the definition of the element - - -
type MatchID2RE_ID_Elem_TY is record
  mid:MatchID; -- KEY
  re:RE_Elem_AC;
  ia:I_A;
  val_J:Integer;
end record;
```

This sums up what MatchPack is doing, it keeps an index of all nodes and a quick way to access individual nodes without reading through the links of the node three.

4.5 PatternPack

The node three from a pattern can be created with a call to a function (A and B) or if a pattern variable already exists then a call of a procedure (C and D)

A) function compileM (s : CFix ; urv : RangeVectorPack.Map_TY.Map := RangeVectorPack.Map_TY.Empty_Map) return Pattern_AC
B) function compileU (s : CFix ; urv : RangeVectorPack.Map_TY.Map := RangeVectorPack.Map_TY.Empty_Map) return Pattern_AC
C) procedure compileM (ptc : in out Pattern_AC ; s : CFix ; urv : RangeVectorPack.Map_TY.Map := RangeVectorPack.Map_TY.Empty_Map)
D) procedure compileU (ptc : in out Pattern_AC ; s : CFix ; urv : RangeVectorPack.Map_TY.Map := RangeVectorPack.Map_TY.Empty_Map)

Note that alternatives B and D does an upper-case translation of the given pattern codepoint-array (CFix) and search argument before any search.
The private type of PatternPack (Pattern) is:

```
private type Pattern is tagged
record
  state: PST_TY;
```

```
  top: RE_Elem_AC;
  pool:JetsTool.Pool_TY;
  uc:boolean;
end record;
```

containing:

- state – defined in Compiler and contain the result of a search, PST_TY (PST_NONE, PST_MATCH, PST_FAIL)
- top – root of node three
- pool – the node three (type Pool_TY is new Controlled with record root: RE_Elem_AC; end record)
- uc – True if conversion to upper-case is used (compileU used)

Should the ‘compileU’ really be used ? The upper-case conversion cannot be trusted if the target is text from natural languages. CompileU can be user for parsing code of a programming language where case is not an issue, ADA, COBOL, Fortran. But if the code is C, C++ then use CompileM.

4.5.1 CompileM & CompileU wrappers of Compiler.CompileP

The compile is done in the Compile package by CompileP procedure. Only operations done in PatternPack package is the change to upper case and storing of the result.

4.5.2 Matches

The function Matches

```
function matches ( pattern : Pattern_AC ; startPos : Integer ; nextPos : out
Integer ; c4match : CFix ; match : out MatchPack.Match_TY ; squ:Integer :=0 )
return Boolean;
```

is also only a wrapper to MatchesI function (page 16) to solve the problem of conversion to upper case if that was requested.

4.5.3 What is getNestElem ?

Function getNestElem is used to synchronize nodes in the node tree with the nest stack during the project. Function definition:
function getNestElem (nest : in out Nest_TY.vector ; currRE : RE_Elem_AC ; resultCursor : out Nest_TY.cursor) return NestElem_TY
containing:

- ✔ nest : in out Nest_TY.vector
- ✔ currRE : RE_Elem_AC
- ✔ resultCursor : out Nest_TY.cursor

returns a nest element.
A remainder how the nest stack is defined in the code:

```
Type NestElem_TY is
record
rec:RE_Elem_AC;
cnt_min:Integer; -- default Integer'First
cnt_max:Integer; -- default Integer'First
c4match_first:Integer; -- No value = Integer'First | update by REC node
c4match_last:Integer; -- No value = Integer'First  | update by REC node
skipCodePoint_candidate:Integer; -- Integer'First, Integer'Last or c4match-index
end record;
```

```
package Nest_TY is new Ada.Containers.Vectors(Natural,NestElem_TY);
```

The work of then function

1) Check last element in stack. If nest stack is empty create a new nest-element, if then last element refers to the same node as defined in the call return this node.
2) We have an empty node to update
   - ✗ create a REQ_CHAR nest-elem – retrieve values for cnt_min and cnt_max from node. Set first and last values.
   - ✗ create a REQ_PERIOD nest-elem – retrieve values for cnt_min and cnt_max from node. Set first and last values.
   - ✗ create a REQ_CLASS nest-elem – retrieve values for cnt_min and cnt_max from node. Set first and last values.
   - ✗ create a REQ_PALCAP nest-elem – retrieve values for cnt_min and cnt_max from node
   - ✗ create a REQ_PAL nest-elem – retrieve values for cnt_min and cnt_max from node
   - ✗ create a REQ_BEGIN nest-elem – set first, last and min values
3) update cnt_min and cnt_max values
4) set skipPoint_candidate to initial value, we don’t skip any codepoints
5) append the new nest element to stack

Where cnt_min and cnt_max refers to how many codepoints is covered by this nest element and c4match-first and -last refers to positions on c4match CFix codepoint array. Values (1,0) is considered to be of null length.

4.5.4 The real matcher, the MatchesI function

The matching is a highly recursive process with following main operators scanning through the links in the node three starts from the root node using function matchesLeftRight.
The target is to build a nest-vector based on the node three and the c4match named CFix literal from startPos forward.

```
Type NestElem_TY is
record
rec:RE_Elem_AC;
cnt_min:Integer; -- default Integer'First
cnt_max:Integer; -- default Integer'First
c4match_first:Integer; -- No value = Integer'First | update by REC node
c4match_last:Integer; -- No value = Integer'First  | update by REC node
skipCodePoint_candidate:Integer; -- Integer'First, Integer'Last or c4match-index
end record;

package Nest_TY is new Ada.Containers.Vectors(Natural,NestElem_TY);
```

the second ordered set which is used is SkipCodePoint set. This solves the problem when there are more than one candidate for the start of the search in the codepoint array (CFix). If the first candidate don’t result in a success, then jump over and try the next.

```
package SkipCodePoint_TY is new Ada.Containers.Ordered_Sets(
Element_Type=> Integer,
"<"=> lessSkipCodePoint,
"="=> equalSkipCodePoint);
```

This feature is backed up by the char_period_class_vector vector in the node and content for char_period_class_vector is generated by the third stage in the Compiler (page 10).
The function:
function matchesI (pattern : Pattern_AC ; startPos : Integer ; nextPos : out Integer ; c4match : CFix ; match : out MatchPack.Match_TY ; squ:Integer := 0) return Boolean
and containing

- pattern : Pattern_AC (page 14) - from where we get the node three (pool : JetsTool.Pool_TY) and the starting point or root (top : RE_Elem_AC)
- startPos : Integer – start index in c4match
- nextPos : out Integer – index in c4match where to start next time
- c4match : CFix – the CFix, the source data which is to be searched
- match : out MatchPack.Match_TY (page 14) - result
- squ:Integer := 0 – how many 'squeeze' (page 21) operation should be done. Default is zero, don't perform any 'squeeze'.

At first the 'match' output variable is created and initialized (MatchPack.Initialize(match);) and then a check is done to be sure that a search can be performed.
An (simplified) algorithm for matchesI would be:

```
while try2match loop
  try2match:=FALSE; ← we suppose that it will be a success or if fail no skipcodepoints exists
  old_out_ia:=out_ia; ← save the index values of c4match
  SkipCodePoint_TY.Clear(skipcodepoint); ← remove all skipcodepoints
  loop ← TRY-AGAIN
    Nest_TY.Clear(nest); ← clear the nest stack for this try
    nestElem:=getNestElem(nest,currRE,resultCursor); ← insert the first element for the nest stack
    rc:=matchesLeftRight(currRE.right,in_ia,out_ia,c4match,nest,skipcodepoint);
    ← Note the skipcodepoint set maybe updated in the call
    if rc = TRI_TRUE then ← if success of the search return
      exit;
    end if;
    if changeSkipcodepoints(nest,skipcodepoint) then ← are there any skipcodepoints left ?
      skipcodepoint_cnt:=skipcodepoint_cnt + 1;
    else
      exit; ← this is a fail
    end if;
  end loop; ← TRY-AGAIN if we found a skipcodepoint
  ←Here is the simplification, in the code we have here the 'squ' or the 'squeeze' handling
end loop; -- while try2match loop
```

If the search is a success then TRUE is returned, else FALSE. The algorithm above also lacks to describe the 'dollar end' problem of a regular expression pattern.
Note the call of matchesLeftRight (page 18).

4.5.4.1 Result to

The matchesI returns boolean but if successful

- nextPos : out Integer
- match : out MatchPack.Match_TY

The return of 'match' is a result of parsing through the nest stack looking for type REQ_PALCAP nodes and inserting I_A values (first and last index value of seached **c4match**) . The a call of MatchPack.getMatch returns to I_A_ARRAY with the index pairs, the Q_A type variables. At last

the application decides what to do with this information, pulling sub codepoint arrays from c4match or using the index values as they are.

4.5.5 Matches Element

Function definition:
function matchesElem (currRE : RE_Elem_AC ; in_ia : I_A ; out_ia : out I_A ; c4match : CFix ; nest : in out Nest_TY.Vector ; skipcodepoint : in out SkipCodePoint_TY.Set) return TRINITY
containing:

- ✔ currRE : RE_Elem_AC – current node of type REQ_CHAR, REQ_PERIOD or REQ_CLASS
- ✔ in_ia : I_A – index of search, first and last codepoint in c4match
- ✔ out_ia : out I_A – result of search, first and last codepoint in c4match
- ✔ c4match : CFix – codepoint array from where the search is done
- ✔ nest : in out Nest_TY.Vector – nest stack
- ✔ skipcodepoint : in out SkipCodePoint_TY.Set – skipcodepoint set

and returning TRINITY (TRI_TRUE, TRI_FALSE and TRI_UNDEF).
MatchesElem works in two stages

(1) match as many codepoints to current node which is required. If fewer than required codepoint is found then return TRI_UNDEF
(2) if the node is defined as
   I. greedy – continue matching codepoints to node and return TRI_TRUE
   II. non-greedy – match as few codepoints to node, we try to find first codepoint which can be matched to next node. Possibly we have to use skipcodepoint vector. Return TRI_TRUE

To determinate if a codepoint is of the requested type function codePoint_candidate is used. In this function we have

(a) REQ_CHAR – the simplest codepoint has to be the same as the value in node
(b) REQ_PERIOD – not even a problem, any value is accepted
(c) REQ_CLASS – two possibilities (c.1) codepoint is in the requested group of codepoints, and (c.2) codepoint is not in the requested group of codepoints. Note that REQ_CLASS node demands case c.1 or c.2 for a successful result

4.5.6 Match Horizontal, MatchesLeftRight

Function definition :
function matchesLeftRight (currRE : RE_Elem_AC ; in_ia : I_A ; out_ia : out I_A ; c4match : CFix ; nest : in out Nest_TY.Vector ; skipcodepoint : in out SkipCodePoint_TY.Set) return TRINITY
containing:

- ✔ currRE : RE_Elem_AC – current node
- ✔ in_ia : I_A – index of search, first and last codepoint in c4match
- ✔ out_ia : out I_A – result of search, first and last codepoint in c4match
- ✔ c4match : CFix – codepoint array from where the search is done
- ✔ nest : in out Nest_TY.Vector – nest stack
- ✔ skipcodepoint : in out SkipCodePoint_TY.Set – skipcodepoint set

and returning TRINITY (TRI_TRUE, TRI_FALSE and TRI_UNDEF).
The function searches from the current node through the node three by using then right link and tries to match nodes to data in c4match. If found data match the nest stack is updated with a new nest-element (NestElem_TY) and process continues with next node in right link. If data don’t fit then we have a fail, TRI_FALSE or TRI_UNDEF.

If the current node is of type

- REQ_PAL or REQ_PALCAP – call matchesPal function (page 19).
  - ✔ at return of TRI_TRUE – Insert returned last index value we got from matchesPal in stack (nest element) and continue to next node if rlink contains a node
  - ✔ at return of TRI_UNDEF – restore nest element to original values and return this function with value TRI_UNDEF
  - ✔ at return of TRI_FALSE – restore nest element to original values and return this function with value TRI_FALSE
- REQ_CHAR, REQ_PERIOD or REQ_CLASS – call matchesElem function (page 18).
  - ✔ at return of TRI_TRUE – Insert returned last index value we got from matchesElem in stack (nest element) and continue to next node if rlink contains a node
  - ✔ at return of TRI_UNDEF – restore nest element to original values and return this function with value TRI_UNDEF
  - ✔ at return of TRI_FALSE – restore nest element to original values and return this function with value TRI_FALSE

if no nodes is found in this rlink-chain then return function as a success (TRI_TRUE).

4.5.7 MatchesPal

Function for handling parenthesized expression in the regular expression. Function definition:
f**unction matchesPal (currRE : RE_Elem_AC ; in_ia : I_A ; out_ia : out I_A ; c4match : CFix ; nest : in out Nest_TY.Vector ; skipcodepoint : in out SkipCodePoint_TY.Set) return TRINITY**
containing:

- ✔ currRE : RE_Elem_AC – current node of type REQ_PAL or REQ_PALCAP
- ✔ in_ia : I_A – index of search, first and last codepoint in c4match
- ✔ out_ia : out I_A – result of search, first and last codepoint in c4match
- ✔ c4match : CFix – codepoint array from where the search is done
- ✔ nest : in out Nest_TY.Vector – nest stack
- ✔ skipcodepoint : in out SkipCodePoint_TY.Set – skipcodepoint set

and returning TRINITY (TRI_TRUE, TRI_FALSE and TRI_UNDEF).
MatchesPal works in two stages

(1) match as many codepoints to current node which is required. If fewer than required codepoint is found then return TRI_UNDEF
(2) if the node is defined as
   I. greedy – continue matching codepoints to node and return TRI_TRUE
   II. non-greedy – match as few codepoints to node, we try to find first codepoint which can be matched to next node. Possibly we have to use skipcodepoint vector. Return TRI_TRUE

For matching codepoints function MatchPalSub (page 19) is called.

4.5.8 MatchesPalSub

The MatchesPalSub works through to vertical chain, from top downwards .Function definition:
function matchesPalSub (currRE : RE_Elem_AC ; in_ia : I_A ; out_ia : out I_A ; c4match : CFix ; nest : in out Nest_TY.Vector ; skipcodepoint : in out SkipCodePoint_TY.Set) return TRINITY
containing the same parameter and -types as MatchesPal (page 19) except currRE can also be of type UPDOWN.
If the vertical chain contains more than one node then function Daisy (page 20) is called else function matchesInternal, also here known as InternalMatch (page 20) is called.

4.5.9 InternalMatch

A wrapper with one purpose to call matchesLeftRight (page 18). Used by Daisy (page 20) and MatchesPalSub (page 19).

4.5.10 The Daisy chain, Daisy function

The function Daisy activates a second tread during handling of the vertical chain.
Function definition:
function daisy(this : RE_Elem_AC ; in_ia : I_A ; out_ia : out I_A ; c4match : CFix ; nest : in out Nest_TY.Vector ; skipcodepoint : in out SkipCodePoint_TY.Set) return TRINITY
containing:

- ✔ currRE : RE_Elem_AC – current node
- ✔ in_ia : I_A – index of search, first and last codepoint in c4match
- ✔ out_ia : out I_A – result of search, first and last codepoint in c4match
- ✔ c4match : CFix – codepoint array from where the search is done
- ✔ nest : in out Nest_TY.Vector – nest stack
- ✔ skipcodepoint : in out SkipCodePoint_TY.Set – skipcodepoint set

and returning TRINITY (TRI_TRUE, TRI_FALSE and TRI_UNDEF).
Nothing strange about this definition.
Then the process splits in two

(a) currRE – follow the down link in matchesPalSub and call daisy is there exists a down link from our down link
(b) currRE.right (this.right) – run matchesInternal with this value

The effect is that every alternative in a parentesed expression (‘|’) is run in its own thread.

NX.Valuate / NX.Result (a)	MatchesInternal (b)
this	this
in_ia	in_ia
/ Hout_ia	Gout_ia
c4match	c4match
Nest / Hnest	Nest
Skipcodepoint / Hskipcodepoint	Skipcodepoint
Result → Hr	Result → Gr

(black = input, red = output, green = in- and output)

After these three statements we have to make a choice:
NX.Valuate(this,in_ia,nest,skipcodepoint);
Gr:=matchesInternal(this,in_ia,Gout_ia,c4match,nest,skipcodepoint);
NX.Result(Hout_ia,Hr,Hnest,Hskipcodepoint);

.

The choice	Gr = TR_TRUE	Gr /= TR_TRUE
Hr = TR_TRUE	Compare Gout_ia and Hout_ia ➢ Gout_ia > Hout_ia - Return TR_TRUE with G-values (and Nest and Skipcodepoint) ➢ Hout_ia > Gout_ia - Return TR_TRUE with H-values ➢ Gout_ia = Hout_ia - Return TR_TRUE with G-values (and Nest and Skipcodepoint)	Return TR_TRUE with H-values
Hr /= TR_TRUE	Return TR_TRUE with G-values (and Nest and Skipcodepoint)	Return TR_UNDEF

The second thread is created with the task NX and started with the Valuate-call and we get the results to main thread through the Result-call. It is important that both thread don't changes values they have in common, that is the reason for the copy-call of Nest_TY and SkipCodePoint_TY.

4.5.11 The squeeze operation

Squeeze is highly experimental and maybe illegal but can still be useful in some cases.
Function definition:
function matchesSqueeze(nest : in out Nest_TY.Vector ; tryCnt : Integer ; c4match : CFix ; rc : TRINITY ; src_need : out Integer) return Boolean
containing:

- ✔ nest : in out Nest_TY.Vector
- ✔ tryCnt : Integer
- ✔ c4match : CFix
- ✔ rc : TRINITY
- ✔ src_need : out Integer

The boolean function matchesSqueeze is used for restarting the search operation if the search failed. The restart code is in matchesI (page) looks like this:

```
case squCnt is
when 0 => null; ← SquCnt variable controls the restarting and default, zero, means no restarting
when others =>
  squCnt:=squCnt – 1; ← !
  src_need_prev:=src_need; ← amount of space which was needed for success
  ← a check is done in function matchesI that previous src_need is not the same as last src_need
  src_need:=0; ← a new value for src_need
  tryCnt:=tryCnt + 1; ← (for debug and info purposes)
  if matchesSqueeze(nest,tryCnt,c4match,rc,src_need) then
    Nest_TY.Clear(nest);
    in_ia:=(startPos,c4match'Last(1) + 1); ← reset start index
    nestElem:=getNestElem(nest,currRE,resultCursor); ← synchronize nest to currRE thread node
```

```
    try2match:=TRUE; ← indicate to matchesI function that a restart is needed
   end if; -- if matchesSqueeze
 end case; -- squ
```

By squeezing the length of non-mandatory node results we try to find space for what is needed. The squeeze is done from that back of the result and as long as there is a need for codepoints.
Nodes of type REC_CHAR, REC_CLASS and REC_PERIOD has a mandatory part and a optional part from where a squeeze can be done. More complicated is nodes of type REC_PAL and REC_PALCAP especially if the node has more than one repetitions and this isn’t fully covered, more work is needed.

5 Utilities

Y2018.Text.Util contains two utilities in it's own subpackage

- Y2018.Text.Util.InFileUTF8 – reading a file assuming that the file contains UTF-8 bytes and returning a codepoint as an entity
- Y2018.Text.Util.UrvPack – mapping of codepoints to codepoint groups

Compiling package Y2018.Text.Util don't need precompilation and is compiled before the precompiler CCompile (page 26) is compiled.

5.1.1 Y2018.Text.Util.InFileUTF8

InFileUTF8 package has a private type, self :

```
private type self is tagged limited
record
  infile:DirectTY.File_Type;
  utf8cnt:Long_Long_Integer;
  bytecnt:Long_Long_Integer;
  reason:RS_TY;
end record;
```

with elements:

- ✔ infile – file type – package DirectTY is new Ada.Direct_IO(Character)
- ✔ utf8cnt – number of codepoints found
- ✔ bytecnt – number of bytes read
- ✔ reason – with values (RS_TY)
  - ✗ RS_NONE – OK
  - ✗ RS_MISS – Last UTF-8 character in a file is missing bytes
  - ✗ RS_ILLEGAL – Value in a byte cannot be an UTF-8 character or a part of an UTF-8 character
  - ✗ RS_EOF – End of File reached

and with procedure and function calls:

- procedure open (base : in out Y2018.Text.Util.InFileUTF8.self ; infilename : String)
- function read (base : in out Y2018.Text.Util.InFileUTF8.self ; result : out CVar ; replace : CodePoint := CodePoint'Val(16#FFFD#) ; action : Boolean : =FALSE) return RS_TY – Two defaulted parameters, (1) 'replace' codepoint value, default to replace-value, and (2) 'action' if true throw an exception else do nothing
- procedure close (base : in out Y2018.Text.Util.InFileUTF8.self)
- function eof (base : Y2018.Text.Util.InFileUTF8.self) return Boolean – return True if end of file reached
- function reason (base : in out Y2018.Text.Util.InFileUTF8.self) return RS_TY
- function utf8cnt (base : in out Y2018.Text.Util.InFileUTF8.self) return Long_Long_Integer
- function bytecnt (base : in out Y2018.Text.Util.InFileUTF8.self) return Long_Long_Integer

The target is to read any file codepoint for codepoint.

5.1.2 Y2018.Text.Util.UrvPack

Package UrvPack has it's own private type

```
private type MapPool is tagged
record
```

```
u:Jets.RangeVectorPack.Map_TY.Map;
end record;
```

Note that RangeVectorPack.Map_TY.Map is defined in RangeVectorPack all tough not used there but here. The reason is the used order of compilation of the project.

The use of UrvPack is described in UrvPack utilities in the Regular Expression in ADA, Y2018-Text-project pages. UrvPack utilities is only that part of codepoint groups which is used in regular expressions in this project. A much wider viewpoint can be found on project pages under heading CreateURV, where you can create DAT-files and combine groups or maps of codepoints in RangeVectorPack.UserRangeVectorMap_TY.Map maps.
Then we have the CreateUnicodeBlock to make it possible to construct DAT-files from blocks of codepoints and CmakeGC_URV to make DAT-files of what is needed directly from Unicodedata.txt.

6 Support for testing

The support is writing to a debug file in package Y2018.Text.TestQ.
The name of the debug file is constructed from expression "q" & Ada.Task_Identification.Image (Ada.Task_Identification.Current_Task) & ".lst" and the target directory is always current directory.
This makes it possible to write debug from all threads without any delays.
The debugging starts always with a call of procedure Qinit (*procedure Qinit(s:String)*) to clear current directory from all files named “q<anything>.lst”. When a thread writes it first debug message then as a prefix a timestamp is written into the corresponding debug file.

If the alternate debug is used the name of the debug file is "q" & Ada.Task_Identification.Image (Ada.Task_Identification.Current_Task) & "_alt.lst". Qinit will also remove these files from the current directory.

To make it easier to spot debug in source code the Qappend and Qappend_alt calls has as first parameter a ANKA type value, if no ANKA type value then default is “KALLE”.

- ✔ KALLE – no prefix
- ✔ KNATTE – one TAB (horizontal tabulator or 16#9#) as prefix
- ✔ TJATTE – two TAB’s as prefix
- ✔ FNATTE – three TABS’s as prefix
- ✔ KAJSA – special marking as prefix for important messages (“!]” and one TAB)
- ✔ KNASE – special marking as prefix for temporary messages (".]" and one TAB)
- ✔ BITTAN – (should never be used because no messages is written to the debug file)

It is important to write the debug on one line and be aware that this line will be removed when the source code is exported from test environment to quality assurance environment, if tools of Y2018.Text project is used. Note also that comments containing debug calls are also removed.

7 Ccompile – The precompiler

It is possible to write code for project Y2018.Text utilities but the result don’t look nice. Ada 2012 don’t support any form of bit-21 codepoint type but uses UTF-8 with variable length characters (one to four bytes).
Preprocessor CCompile looks for entities in the source code and replaces these with static codepoint array (CFix) literals.

```
patternProcedureName_C:constant Cfix:=Y2018.Text.Core.UTF.To21(
"^[\s]*PROCEDURE[\s]+([\w\.]+)");

patternPackageHeadName_C:constant Cfix:=Y2018.Text.Core.UTF.To21(
"^[\s]*PACKAGE[\s]+([\w\.]+)");

patternPackageBodyName_C:constant Cfix:=Y2018.Text.Core.UTF.To21(
"^[\s]*PACKAGE[\s]+BODY[\s]+([\w\.]+)");

patternDsect_C:constant Cfix:=Y2018.Text.Core.UTF.To21(
"^[\s]*--[\s]*WITH[\s]+DSECT[\s]*;");
```

Lines are read and :

(1) If line contains pattern patternProcedureName_C then this file contains a procedure (.adb). If needed a package name & "_PSECT" will be generated for static literals
(2) if line contains pattern patternPackageHeadName_C then this file contains a package head (.ads). If needed a package name & "_HSECT" will be generated for static literals
(3) if line contains pattern patternPackageBodyName_C then this file contains a package body (.adb). If needed a package name & "_BSECT" will be generated for static literals
(4) if line contains pattern patternDsect_C then remember this line for the possible with-definition for the new result file. If this pattern is not found before any of the tree cases above the with-definition is generated in the new result file just above the found procedure or package definition.

The preprocessor reads the input file and returns one or two output files with the substitutions needed.
Lines from input is checked with this pattern

```
patternLineSimple_C:constant CFix:=Y2018.Text.Core.UTF.To21(
"^(.*?)(" &
"--|" & - - comment (1)
"""(?:[^""]|""""")*"".|" & - - possible a c-literal of CFix type (2)
"'\\[BFNRT]'C|" & - - a codepoint literal (3)
"'.'C" &")"); - -  a codepoint literal (3)
```

The line is searched as long as new result is not found

(1) rest of the line is comment
(2) if last character is a ‘c’ then a CFix type literal is found and a static codepoint array must be generated in the PSECT, HSECT or BSECT result file for this literal and the literal found in the input must be replaced in the output
(3) replace the value with a CodePoint'Val(16#….#) value.

If a line with case (2), a CFix literal, is not found then the PSECT, HSECT or BSECT result file is not generated.

7.1.1 Why these PSECT, HSECT or BSECT files

The reason for this solution is that when the precompiler makes it’s search the precompiler is not aware of possible companion adb- or ads-file. There exists a drawback to this solution that you cannot refer to a CFix literal in the companion adb- or ads-file.

8 Conclusions

At *Regular Expression in ADA, Y2018-Text-project* page a number of tools are mention

(a) Chesslist – example and showcase of what can be done with InFileUTF8 (page 23)
(b) Classlist – list the content of one or more DAT-files
(c) Cmakegc_urv – make General Category URV-maps from Unicode.org UnicodeData.txt (URV page 23)
(d) CodeRewrite – utility for rewriting source code when changing code from TEST to Quality Assurance (QA)
(e) CreateGC – tool for generating Y2018.Text.Core.GC from Unicode.org UnicodeData.txt file
(f) CreateULTcase – tool for generating Y2018.Text.Core.ULTcase from Unicode.org UnicodeData.txt file
(g) CreateUnicodeblock – create a DAT-file based on which block a codepoint is a member of (URV page 23)
(h) CreateURV – create DAT-files for storing RangeVectorPack.UserRangeVectorMap_TY.Map's (URV page 23 and RangeVectorMap page 8)

and a number of examples on the same project pages.